Why Containers and WebAssembly Work Well Together

Tyler Charboneau

Editor's note: The Docker+Wasm Technical Preview is now available. Find out more about the preview and try it for yourself!

—

Developers favor the path of least resistance when building, shipping, and deploying their applications. It's one of the reasons why containerization exists — to help developers easily run cross-platform apps by bundling their code and dependencies together.

While we've built upon that with Docker Desktop and Docker Hub, other groups like the World Wide Web Consortium (W3C) have also created complementary tools. This is how WebAssembly (AKA "Wasm") was born.

Though some have asserted that Wasm is a replacement for Docker, we actually view Wasm as a companion technology. Let's look at WebAssembly, then dig into how it and Docker together can support today's demanding workloads.

What is WebAssembly?

WebAssembly is a compact binary format for packaging code to a portable compilation target. It leverages its JavaScript, C++, and Rust compatibility to help developers deploy client-server web applications. In a cloud context, Wasm can also access the filesystem, environment variables, or the system clock.

Wasm uses modules — which contain stateless, browser-compiled WebAssembly code — and host runtimes to operate. Guest applications (another type of module) can run within these host applications as executables. Finally, the WebAssembly System Interface (WASI) brings a standardized set of APIs to enable greater functionality and access to system resources.

Developers use WebAssembly and WASI to do things like:

- Build cross-platform applications and games
- Reuse code between platforms and applications
- Running applications that are Wasm and WASI compilable on one runtime
- Compile WebAssembly files to a single target for dependencies and code

How does WebAssembly fit into a containerized world?

If you're familiar with Docker, you may already see some similarities. And that's okay! Matt Butcher, CEO of Fermyon, explained how Docker and Wasm can unite to achieve some pretty powerful development outcomes. Given the rise of cloud computing, having multiple ways to securely run any software atop any hardware is critical. That's what makes virtualized, isolated runtime environments like Docker containers and Wasm so useful.

Matt highlights Docker Co-Founder Solomon Hykes' original tweet on Docker and Wasm, yet is quick to mention Solomon's follow-up message regarding Wasm. This sheds some light on how Docker and Wasm might work together in the near future:

Accordingly, Docker and Wasm can be friends — not foes — as cloud computing and microservices grow more sophisticated. Here's some key points that Matt shared on the subject:

Let Use Cases Drive Your Technology Choices

We have to remember that the sheer variety of use cases out there far exceeds the capabilities of any one tool. This means that Docker will be a great match for some applications, WebAssembly for others, and so on. While Docker excels at building and deploying cross-platform cloud applications, Wasm is well-suited to portable, binary code compilation for browser-based applications.

Developers have long favored WebAssembly while creating their multi-architecture builds. This remains a sticking point for Wasm users, but the comparative gap has been narrowed with the launch of Docker Buildx. This helps developers achieve very similar results as those using Wasm. You can learn more about this process in our recent blog post.

During his presentation, Matt introduced what he called "three different categories of compute infrastructure. Each serves a different purpose, and has unique relevance both currently and historically:

- **Virtual machines (heavyweight class)** – AKA the "workhorse" of the cloud, VMs package together an entire operating system — kernels and drivers included, plus code or data — to run an application virtually on compatible hardware. VMs are also great for OS testing and solving infrastructure challenges related to servers, but, they're often multiple GB in size and consequently start up very slowly.
- **Containers (middleweight class)** – Containers make it remarkably easy to package all application code, dependencies, and other components together and run cross-platform. Container images measure just tens to hundreds of MB in size, and start up in seconds.
- **WebAssembly (lightweight class)** – A step smaller than containers, WebAssembly binaries are minuscule, can run in a secure sandbox, and start up nearly instantly since they were initially built for web browsers.

Matt is quick to point out that he and many others expected containers to blow VMs out of the water as the next big thing. However, despite Docker's rapid rise in popularity VMs have kept growing. There's no zero-sum game when it comes to technology. Docker hasn't replaced VMs, and similarly, WebAssembly isn't poised to displace the containers that came before it. As Matt says, "each has its niche."

Industry Experts Praise Both Docker and WebAssembly

A recent New Stack article digs into this further. Focusing on how WebAssembly can replace Docker is "missing the point," since the main drivers behind these adoption decisions should be business use cases. One important WebAssembly advantage revolves around edge computing. Because Docker containers are now working more and more harmoniously with edge use cases. For example, exciting IoT possibilities await, while edge containers can power streaming, real-time process, analytics, augmented reality, and more.

If we reference Solomon's earlier tweet, he alludes to this when envisioning Docker running Wasm containers. The trick is identifying which apps are best suited for which technology. Applications that need heavy filesystem control and IO might favor Docker. The same applies if they need sockets layer access. Meanwhile, Wasm is optimal for fuss-free web server setup, easy configuration, and minimizing costs.

With both technologies, developers are continuously unearthing both new and existing use cases.

Docker and Wasm Team Up: The Finicky Whiskers Game

Theoretical applications are promising, but let's see something in action. Near the end of his talk, Matt revealed that the Finicky Whiskers game he demoed to start the session actually leveraged Docker, WebAssembly, and Redis. These three technologies comprised the game engine to form a lightning-fast backend.

Matt's terminal displays backend activity as he interacts with the game.

Finicky Whiskers relies on eight separate WebAssembly modules, five of which Matt covered during his session. In this example, each button click sends an HTTP request to 5pin — Fermyon's framework for running web apps, microservices, and server applications.

These clicks generate successively more Wasm modules to help the game run. These modules spin up or shut down almost instantly in response to every user action. The total number of invocations changes with each module. Modules also grab important files that support core functionality within the application. Though masquerading as a game, Finicky Whiskers is actually a load generator.

A Docker container has a running instance of Redis and pubsub, which are used to broker messages and access key/value pairs. This forms a client-server bridge, and lets Finicky Whiskers communicate. Modules perform data validation before pushing it to the Redis pubsub implementation. Each module can communicate with the services within this Docker container — along with the file server — proving that Docker and Wasm can jointly power applications:

Specifically, Matt used Wasm to rapidly start and stop his microservices. It also helped these services perform simple tasks. Meanwhile, Docker helped keep the state and facilitate communication between Wasm modules and user input. It's the perfect mix of low resource usage, scalability, long-running pieces, and load predictability.

Containers and WebAssembly are Fast Friends, Not Mortal Enemies

As we've demonstrated, containers and WebAssembly are companion technologies. One isn't meant to defeat the other. They're meant to coexist, and in many cases, work together to power some pretty interesting applications. While Finicky Whiskers wasn't the most complex example, it illustrates this point perfectly.

In instances where these technologies stand apart, they do so because they're supporting unique workloads. Instead of declaring one technology better than the other, it's best to question where each has its place.

We're excited to see what's next for Wasm at Docker. We also want Docker to lend a helping hand where it can with Wasm applications. Our own Jake Levine, Head of Product, says it best:

> *"Wasm is complementary to Docker — in whatever way developers choose to architect and implement parts of their application, Docker will be there to support their development experience," Levirne said. Development, testing and deployment toolchains that use Docker make it easier to maintain reproducible pipelines for application delivery regardless of application architecture, Levirne said. Additionally, the millions of pre-built Docker images, including thousands of official and verified images, provide "a backbone of core services (e.g. data stores, caches, search, frameworks, etc.)" that can be used hand-in-hand with Wasm modules."*

We even maintain a collection of WebAssembly/Wasm images on Docker Hub! Download Docker Desktop to start experimenting with these images and building your first Docker-backed Wasm application. Container runtimes and registries are also expanding to include native WebAssembly support.

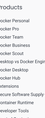
Posted
Jul 1, 2022

Post Tags
containers
Docker Desktop
Docker Hub
Wasm
WebAssembly

Categories
Community
Company
Engineering
Products

Tags: Containers, Docker Desktop, Docker Hub, Wasm, WebAssembly

Docker State of Application Development Survey 2022: Share Your Thoughts on Development
By Jake Levirne October 20, 2022



Signing Docker Official Images Using OpenPubkey
By Jonny Dyson October 13, 2022

Getting Started with JupyterLab as a Docker Extension
By Marcelo Ochoa October 12, 2022